佑 華 微 電 子 股 份 有 限 公 司

# AMCA

# User Manual


*佑華微電子股份有限公司*

新竹市光復路二段 295 號 9 樓之 1 號

電話: 03-573 6660

傳真: 03-573 6661

www..ealpha.com.tw

***Alpha Microelectronics Corp.***

9F-1, 295, Sec. 2, Kuang Fu Rd., Hsinchu, Taiwan

Tel : +886-3-573 6660

Fax: +886-3-573 6661

www.ealpha.com.tw



Rev 2.1 2004/2/12

# Table of Contents

**1. General Information ..... 3**
1.1 About This Guide ..... 3
1.2 Recommended Reading ..... 4
1.3 The ALPHA Internet Web Site ..... 5
1.4 Development Systems Customer Notification Service ..... 5
1.5 Customer Support ..... 5

**2. AMCA Preview ..... 6**
2.1 What AMCA Is ..... 6
2.2 What AMCA Does ..... 6
2.3 Compatibility Issues ..... 6

**3. AMCA – Installation and Getting Started ..... 7**
3.1 Installation ..... 7
3.2 Overview of Assembler ..... 7
3.3 Assembler Input/Output Files ..... 8

**4. Using AMCA with Windows ..... 12**
4.1 Windows Shell Interface ..... 12

**5. Directive Language ..... 14**
5.1 Highlights ..... 14
5.2 Directive Summary ..... 14
5.3 BREAK – Jump Out Point in a Logic Block ..... 16
5.4 CASE – Define an Option Item of SWITCH ..... 17
5.5 CBLOCK – Define a Block of Constants ..... 18
5.6 CONSTANT – Declare Symbol Constant ..... 19
5.7 CONTINUE – Ignore Statements Afterward and Start Next Loop ..... 19
5.8 DATA – Create Numeric and Text Data ..... 20
5.9 DB – Declare Data of One Byte ..... 21
5.10 DEFAULT – Define an Unconditional Item of SWITCH ..... 21
5.11 #DEFINE – Define a Text Substitution Label ..... 22
5.12 DN – Declare Data of One Nibble ..... 22
5.13 DT – Define Table ..... 23
5.14 DW – Declare Data of One Word ..... 23
5.15 ELSE – Begin Alternative Assembly Block to IF ..... 24
5.16 END – End Program Block ..... 24
5.17 ENDC – End an Automatic Constant Block ..... 24
5.18 ENDFOR – End a For Loop ..... 25
5.19 ENDIF – End Conditional Assembly Block ..... 25
5.20 ENDM – End a Macro Definition ..... 25
5.21 ENDS – Coding Convenience ..... 26
5.22 ENDSW – End a Switch Block ..... 26
5.23 ENDW – End a While Loop ..... 26
5.24 EQU – Define an Assembler Constant ..... 27
5.25 ERROR – Issue an Error Message ..... 27
5.26 EXITM – Exit from a Macro ..... 28
5.27 EXPAND – Expand Macro Listing ..... 28

5.28 FOR – Perform For Loop While Iterator Meets the Condition ..... 28
5.29 IF – Begin Conditionally Assembled Code Block ..... 29
5.30 IFDEF – Execute If Symbol has Been Defined ..... 30
5.31 IFNDEF – Execute If Symbol has not Been Defined ..... 30
5.32 #INCLUDATA – Include Binary Data File ..... 31
5.33 #INCLUDE – Include Additional Source File ..... 31
5.34 LINES – Insert Listing Page Eject ..... 32
5.35 LIST – Listing Options ..... 32
5.36 LOCAL – Declare Local Macro Variable ..... 33
5.37 MACRO – Declare Macro Definition ..... 34
5.38 MAXMACRODEPTH – Define Maximum Macro Depth ..... 34
5.39 MESSG – Create User Defined Message ..... 35
5.40 NEWPAGE – Insert Listing Page Eject ..... 35
5.41 NOEXPAND – Turn off Macro Expansion ..... 36
5.42 ORG – Set Program Origin ..... 36
5.43 RADIX – Specify Default Radix ..... 36
5.44 REPEAT – Begin a Repeat-Until Loop Block Definition ..... 37
5.45 RES – Reserve Memory ..... 37
5.46 SUBTITLE – Specify Program Subtitle ..... 38
5.47 SWITCH – Begin Conditional Switching Assembly Block ..... 38
5.48 TITLE – Specify Program Title ..... 39
5.49 #UNDEFINE – Delete a Substitution Label ..... 39
5.50 UNTIL – Perform Loop Until Condition is True ..... 40
5.51 VARIABLE – Declare Symbol Variable ..... 41
5.52 WHILE – Perform Loop While Condition is True ..... 41

**6. Macro Language ..... 43**

6.1 Macro Syntax ..... 43
6.2 Macro Directives ..... 43
6.3 Text Substitution ..... 44
6.4 Macro Usage ..... 44

**7. Expression Syntax and Operation ..... 45**

7.1 Text Strings ..... 45
7.2 Numeric Constants and Radix ..... 46
7.3 High/Mid/Low ..... 48
7.4 Increment/Decrement (++/--) ..... 48

**Appendix A. Quick Reference ..... 49**

A.1 AMCA Quick Reference ..... 49
A.2 4-Bit MCU List ..... 54
A.3 8-Bit MCU List ..... 55
A.4 4-Bit MCU Instruction Set ..... 56

**Appendix B. AMCA Errors/Warnings ..... 58**

B.1 Errors ..... 58
B.2 Warnings ..... 63

**Glossary ..... 64**

C.1 Terms ..... 64

# 1. General Information

This first chapter contains general information that will be useful to know before working with AMCA.

The information you will garner from this chapter:

• About This Guide

• Recommended Reading

• The ALPHA Internet Web Site

• Development Systems Customer Notification Service

• Customer Support

## 1.1 About This Guide

### 1.1.1 Document Layout

This document describes how to use AMCA to develop code for ALPHA micro-controller applications. The user's guide layout is as follows:

• **Chapter 2: AMCA Preview** – defines AMCA and describes what it does and how it works with other tools.

• **Chapter 3: AMCA** – **Installation and Getting Started** – describes how to install AMCA and gives an overview of operation.

• **Chapter 4: Using AMCA with WIndows®** – describes how to use AMCA with Microsoft Windows via a Windows shell interface.

• **Chapter 5: Directive Language** – describes the AMCA programming language including statements, operators, variables, and other elements.

• **Chapter 6: Macro Language** – describes how to use AMCA's built-in macro processor.

• **Chapter 7: Expression Syntax and Operation** – provides guidelines for using complex expressions in AMCA source files.

• **Appendix A: Quick Reference** – lists ALPHA MCU device instruction sets, hexadecimal to decimal conversions, and ASCII Character Set.

• **Appendix B: AMCA Errors/Warnings** – contains a descriptive list of the errors, and warnings generated by AMCA.

• **Glossary** – A glossary of terms used in this guide.

### 1.1.2 Conventions Used in This Guide

This manual uses the following documentation conventions:

• **Documentation Conventions**

• **Description Represents Examples**

• **Code**

Courier Font User-entered code or sample code

```
#define BITWIDTH
```

Angle Brackets: <>

Variables. Text you supply `<label>, <exp>`

Curly Brackets and Pipe Character:{|}

Choice of mutually exclusive arguments; an OR selection `errorlevel { 0 | 1 }`

Lowercase Characters in Quotes

Type of data `"filename"`

Ellipses: ...

Used to imply, but not show, additional text that is not relevant to the example.

`List ["list_option", ...,"list_option"]`

0xnnnn 0xnnnn represents a hexadecimal number where n is a hexadecimal digit `0xFFFF, 0x003B`

### 1.1.3 Updates

All documentation becomes dated, and this user's guide is no exception. Since AMCA, and other ALPHA tools are constantly evolving to meet customer needs, some actual dialogs and/or tool descriptions may differ from those in this document. Please refer to our web site to obtain the latest documentation available.

## 1.2 Recommended Reading

This user's guide describes how to use AMCA. The user may also find the data sheets for specific micro-controller devices informative in developing firmware.

• **RevisionHistory.TXT**

For the latest information on using AMCA, read the REVISIONHISTORY files (ASCII text files) included with the AMCA software. The REVISIONHISTORY files contain update information that may not be included in this document.

• **Interface**

In-text Bold Characters Designates a button **OK, Cancel**

Uppercase Characters in Angle Brackets: < > Delimiters for special keys. <TAB>, <ESC>

• **Microsoft Windows Manuals**

This manual assumes that users are familiar with Microsoft Windows operating system. Many excellent references exist for this software program, and should be consulted for general operation of Windows.

## 1.3 The ALPHA Internet Web Site

ALPHA provides on-line support on the ALPHA World Wide Web (WWW) site. The web site is used by ALPHA as a means to make files and information easily available to customers. To view the site, the user must have access to the Internet and a web browser, such as Netscape® Navigator or Microsoft® Internet Explorer®.

• **Connecting to the ALPHA Internet Web Site**

The ALPHA website is available by using your favorite Internet browser to attach to:

**http://www.ealpha.com.tw**

The website provides a variety of services. Users may download files for the latest Development Tools, Data Sheets, Application Notes, User's Guides, and Articles. Other data available for consideration is:

• Latest ALPHA Press Releases

• Product Information

• Job Postings

## 1.4 Development Systems Customer Notification Service

ALPHA provided the customer notification service to help our customers keep current on ALPHA products with the least amount of effort. You will receive email notification whenever we change, update, revise or have errata related to that product family or development tool.

## 1.5 Customer Support

Users of ALPHA products can receive assistance through several channels:

• Distributor or Representative

• Field Application Engineer (FAE)

• Hot line

Customers should call their distributor, representative, or field application engineer (FAE) for support.

# 2. AMCA Preview

This chapter will give you a preview of AMCA and its capabilities.

Topics covered in this chapter:

- What AMCA Is
- What AMCA Does
- Migration Path
- Compatibility Issues

## 2.1 What AMCA Is

AMCA Windows-based PC application that provides a platform for developing assembly language code for ALPHA's microcontroller (MCU) families. Use of the ALPHA AMCA Universal Assembler requires an IBM PC/AT® or compatible computer, Microsoft® Windows 98/NT/2000/XP. AMCA supports all MCU products from ALPHA.

## 2.2 What AMCA Does

AMCA provides a universal solution for developing assembly code for all of ALPHA's 8-bit and 4-bit micro-controllers. Notable features include:

- All MCU Instruction Sets
- Window Interfaces
- Rich Directive Language

## 2.3 Compatibility Issues

AMCA is compatible with all ALPHA development systems currently in production. This includes EzCode, AMCIDE, EzSpeech, T2E4 and T2E5. AMCA supports a clean and consistent method of specifying radix (see Chapter 4). You are encouraged to develop new code using the methods described within this document, even though certain older syntaxes may be supported for compatibility reasons.

# 3. AMCA – Installation and Getting Started

This chapter provides instructions for installation of AMCA on your system, and an overview of the assembler (AMCA).

Topics covered in this chapter:

• Installation

• Overview of Assembler

• Assembler Input/Output Files

## 3.1 Installation

Current version of AMCA is for Windows 98/NT/2000/XP version, `AMCA.EXE` has a Windows GUI interface. `AMCA.EXE` may be used with Windows 98/NT/2000/XP. You can obtain AMCA and supporting files from our website. AMCA will be in a zip file. To install:

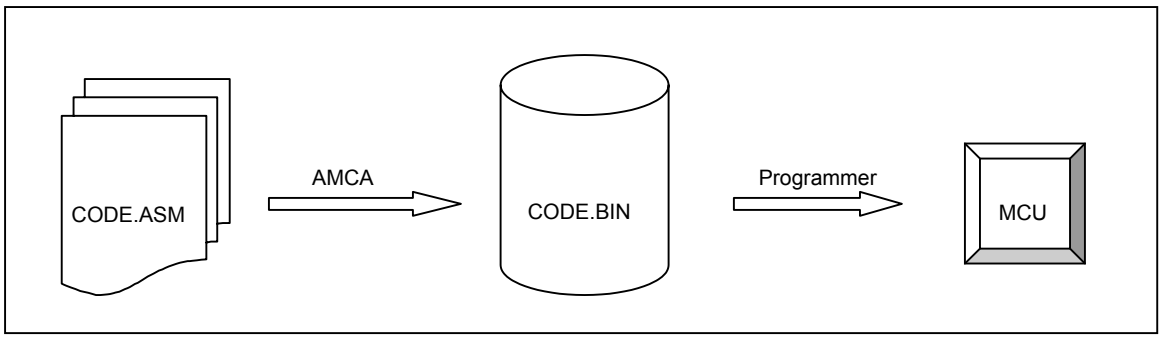
• Create a directory in which to place the files

• Unzip the AMCA files using either WinZip®

## 3.2 Overview of Assembler

AMCA can be used to generate absolute code that can be executed directly by a micro-controller. Absolute code is the default output from AMCA. This process is shown in Figure 3.1. When a source file is assembled in this manner, all values used in the source file must be defined within that source file, or in files that have been explicitly included. If assembly proceeds without errors, a BIN file will be generated, containing the executable machine code for the target device. This file can then be used in conjunction with a device programmer to program the micro-controller.

**Figure 3.1: Generating Absolute Code**



## 3.3 Assembler Input/Output Files

These are the default file extensions used by AMCA and the associated utility functions.

**Table 3.1: AMCA Default Extensions**

| *Extension* | Purpose |
|---|---|
| .ASM | Default source file extension input to AMCA:<br><source_name>.ASM |
| .LST | Default output extension for listing files generated by AMCA:<br><source_name>.LST |
| TMP | Output extension from AMCA for storing the 1st parsing job of AMCA in an assembling session. The 2nd parsing job is based on the content of this file:<br><source_name>.TMP |
| .ERR | Output extension from AMCA for Warning/error files:<br><source_name>.ERR |
| BIN | Output extension from AMCA for the machine code of an application program in binary form:<br><source_name>.BIN |
| .HEX | Output extension from AMCA for representing BIN file in hexadecimal form:<br><source_name>.HEX |
| .DBG | Output extension from AMCA for the symbol and debug file. This file is created for AMCIDE debug mode:<br><source_name>.DBG |

### 3.3.1 Source Code Format (.ASM)

The source code file may be created using any ASCII text file editor. It should conform to the following basic guidelines. Each line of the source file may contain up to four types of information:

- labels
- mnemonics
- operands
- comments

The order and position of these are important. Labels must start in column one. Mnemonics may start in column two or beyond. Operands follow the mnemonic. Comments may follow the operands, mnemonics or labels, and can start in any column. The maximum column width is 255 characters. Blank(s) or a colon must separate the label and the mnemonic, and the mnemonic and the operand(s). Multiple operands must be separated by a comma. For example:

**Example 3.1: Sample AMCA Source Code (Shows multiple operands)**

```
; sample APLHA assembler source code
;
list   p=am8aa000x , c=off ,r=hex
ORG_OFF            equ     0x30
```

```
ORG_SUBOFF      equ     0x00
SUBPPTRADDR     equ     ORG_SUBOFF+ORG_OFF
#include   "2102.h"
org   0x10
movia      0x20
jmp   start
org   0x30
start:
movia       0x30
movar       0x12
end
```

#### 3.3.1.1 Labels

A label must start in column 1. It may be followed by a colon (:), space, tab or the end of line. Labels must begin with an alphabetic character or an under bar (_) and may contain alphanumeric characters, the under bar and the '@' symbol. By default they are case sensitive, but case sensitivity may be overridden by a command line option. If a colon is used when defining a label, it is treated as a label operator and not part of the label itself.

#### 3.3.1.2 Mnemonics

Assembler instruction mnemonics, assembler directives and macro calls can begin in any column. If there is a label on the same line, instructions must be separated from that label by a colon, or by one or more spaces or tabs.

#### 3.3.1.3 Operands

Operands must be separated from mnemonics by one or more spaces, or tabs. Multiple operands must be separated by commas.

#### 3.3.1.4 Comments

AMCA treats anything after a semicolon as a comment. All characters following the semicolon are ignored through the end of the line. String constants containing a semicolon are allowed and are not confused with comments.

### 3.3.2 Listing File Format (.LST)

**Example 2.2: Sample AMCA Listing File (.LST)**

```
Alpha Microelectronics Corp.

AMCA 2.12 Copyright(c) Alpha Microelectronics Corp. [Build:Aug 29 2003]
```

```
File=E:\MyProjects\Build\asm\amca\Sample\AMCASample.asm Date=2003/10/2, 06:22:21 pm

ADDR      OPCODE/VALUE     LINE      TAG SOURCE STATEMENT                          PAGE:1

                           0-0001    ; sample APLHA assembler source cod
                           0-0002    ;
                           0-0003    list   p=am8ba000x , c=off ,r=hex
                000000030  0-0004    ORG_OFF            equ    0x30
                000000000  0-0005    ORG_SUBOFF         equ    0x00
                000000030  0-0006    SUBPPTRADDR        equ    ORG_SUBOFF+ORG_OFF
                           0-0007    #include   "2102.h"
                           1-0001    ;
                000000010  0-0008    org   0x10
000010          D020       0-0009    movia      0x20
000011          6030       0-0010    jmp   start
                000000030  0-0011    org   0x30
                000000030  0-0012    start:
000030          D030       0-0013    movia      0x30
000031          0112       0-0014    movar      0x12
                           0-0015    end


AMCA 2.12 Copyright(c) Alpha Microelectronics Corp. [Build:Aug 29 2003]
File=E:\MyProjects\Build\asm\amca\Sample\AMCASample.asm Date=2003/10/2, 06:22:21 pm

SYMBOL TABLE              TYPE     VALUE                                        PAGE:2

__AM8BA000X                Constant 00000001
ORG_OFF                    Constant 00000030
ORG_SUBOFF                 Constant 00000000
start                      Label    00000030
SUBPPTRADDR                Constant 00000030

SOURCE FILE TABLE
000 E:\MyProjects\Build\asm\amca\Sample\AMCASample.asm
001 E:\MyProjects\Build\asm\amca\Sample\2102.h

PROCESSOR      = AM8BA000X (8 bits)
PROGRAM ROM    = 0x00000000 - 0x0000FFFF
DATA ROM       = 0x00000000 - 0x00FFFFFF
SRAM / SFR     = 0x00000000 - 0x000000FF
```

The listing file format produced by AMCA is straightforward:

The product name and version, the assembly date and time, and the page number appear at the top of every page. The first column of numbers contains the base address in memory where the code will be placed. The second column displays the 32-bit value of any symbols created with the EQU, VARIABLE, CONSTANT, or CBLOCK directives. The third column is reserved for the machine instruction. This is the code that will be executed by the ALPHA MCU. The fourth column lists the associated source file line number for this line. The remainder of the line is reserved for the source code line that generated the machine code. Errors, warnings, and messages are embedded between the source lines, and pertain to the following source line. The symbol table lists all symbols defined in the program.

### 3.3.3 Error File Format (.ERR)

AMCA by default generates an error file. This file can be useful when debugging your code. The format of the messages in the error file is:

```
[<type>] <file> (<line>) <number> <description>
```

**For example:**

```
[Error] C:\PROG.ASM 7 (133) w001: Symbol not previously defined (start)
```

Appendix C describes the error messages generated by AMCA.

### 3.3.4 Hex File Formats (.HEX)

AMCA is capable of producing different hex file formats.

### 3.3.5 Symbol and Debug File Format (.DBG)

When AMCA is evoked by AMCIDE, it produces a DBG file for use in ICE debugging of code.

# 4. Using AMCA with Windows

This chapter is dedicated to describing the version of AMCA for Windows. This version may be run stand-alone as a Windows shell, or within other ALPHA development tools e.g. EzCode and AMCIDE.

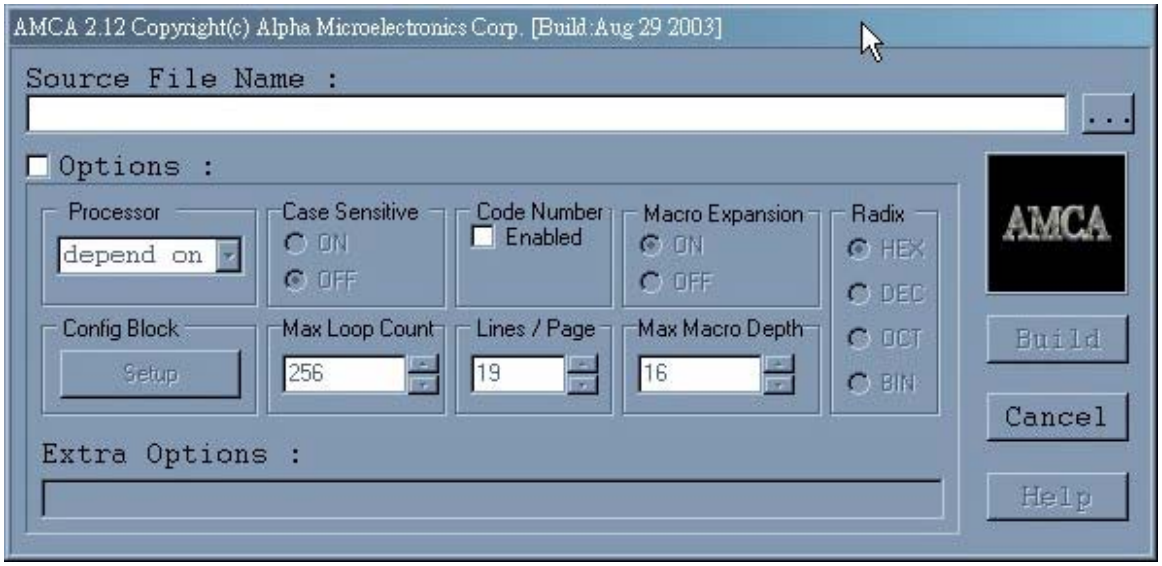
Topics covered in this chapter:

• Windows Shell Interface

## 4.1 Windows Shell Interface

AMCA for Windows provides a graphical interface for setting assembler options. It is invoked by executing `AMCA.EXE` while in Windows.

**Figure 4.1: AMCA Windows Shell Interface**




Select a source file by typing in the name or using the **Browse** button. Set the various options as described below. Then click **Build** to assemble the source file.

**Note:** When AMCA for Windows is invoked through other ALPHA development tools, the options screen is not available. Options are passed from specific tools according to the created ASM file.

**Table 4.1: Assembler Options**

| Option | Usage |
|---|---|
| Radix | Override any source file radix settings |
| Macro Expansion | Override any source file macro expansion settings. |
| Code Number | Produce two extra files (O file and R file) if user checks “Code Number Enabled” box and typed in a three-digital number. |
| Case Sensitivity | Enable/disable case sensitivity. |
| Processor | Override any source file processor settings. |

| Option | Usage |
|---|---|
| Max Macro Depth | Give the max macro depth; Default value is 16 and maximum is 256. |
| Lines/Page | Override any source file lines per page settings; Default is 19, minimum is 10, and maximum is 1000. |
| Max Loop Count | Override any source file maximum loop count settings. Default and maximum are 256. |
| Config Block | Popup a setup window for hardware configuration. |

**Table 4.2: Assembler Extra Options:**

| Extra Option | Usage |
|---|---|
| /unlockrsvmem | For 4-bit MCU only. Allow the programming right in reserved memory area |
| /nocfgblk | For 4-bit MCU only. Ignore the assembly time check for the existence of configuration block file |

# 5. Directive Language

This chapter describes the AMCA directive language. Directives are assembler commands that appear in the source code but are not translated directly into opcodes. They are used to control the assembler: its input, output, and data allocation.

## 5.1 Highlights

There are five basic types of directives provided by AMCA:

• Control Directives – Control directives permit sections of conditionally assembled code.

• Data Directives – Data Directives are those that control the allocation of memory and provide a way to refer to data items symbolically, that is, by meaningful names.

• Listing Directives – Listing Directives are those directives that control the AMCA listing file format. They allow the specification of titles, pagination, and other listing control.

• Macro Directives – These directives control the execution and data allocation within macro body definitions.

• Object File Directives – These directives are used only when creating an object file.

## 5.2 Directive Summary

Table 5.1 contains a summary of directives supported by AMCA. The remainder of this chapter is dedicated to providing a detailed description of the directives supported by AMCA. Each definition will show:

• Syntax

• Description

• Example

**Table 5.1: Directive Summary**

| Directive | Description | Syntax |
|---|---|---|
| BREAK | Escape from a **FOR**, **WHILE** or **REPEAT-UNTIL** loop, or Jump to the end of a **SWITCH** block | break [<Boolean expression>] |
| CASE | Part of a **SWITCH** block; must use **CASE** with **SWITCH** | switch <expression><br>case <expression 1>[,<expression 2>]<br><statements> |
| CBLOCK | Define a block of constants | cblock [<expr>] |
| CONSTANT | Declare symbol constant | constant <label>[=<expr>,...,<label>[=<expr>] ] |
| CONTINUE | Jump to the begin of **FOR**, **WHILE** or **REPEAT-UNTIL** loop that contains **CONTINUE** directive.<br><br>All statements behind **CONTINUE** in a loop are ignored. | continue [<Boolean expression>] |

| Directive | Description | Syntax |
|---|---|---|
| DATA | Create numeric and text data | [<label>] data <expr>,[,<expr>,...,<expr>]<br>[<label>] data "<text_string>"[,"<text_string>",...] |
| DB | Declare data of one byte | [<label>] db <expr>[,<expr>,...,<expr>]<br>[<label>] db "<text_string>"[,"<text_string>",...] |
| DEFAULT | Part of a **SWITCH** block; must use **DEFAULT** with **SWITCH.**<br>Begin default assembly block to **SWITCH** | default<br><statements> |
| #DEFINE | Define a text substitution label | #define <name> [<value>]<br>#define <name> [<arg>,...,<arg>] |
| DN | Declare data of one nibble | [<label>] dn <expr>,[,<expr>,...,<expr>] |
| DT | Define table | [<label>] dt <expr>[,<expr>,...,<expr>]<br>[<label>] dt "<text_string>"[,"<text_string>",...] |
| DW | Declare data of one word | [<label>] dw <expr>[,<expr>,...,<expr>]<br>[<label>] dw "<text_string>"[,"<text_string>",...] |
| ELSE | Begin alternative assembly block to **IF** | else<br><statements> |
| END | End program block | end |
| ENDC | End an automatic constant block | endc |
| ENDFOR | End a **FOR** loop | endfor |
| ENDIF | End conditional assembly block | endif |
| ENDM | End a macro definition | endm |
| ENDS | Directive for coding convenience: presenting **ENDFOR, ENDW, ENDSW, ENDIF** | ends |
| ENDSW | End conditional switching assembly block | endsw |
| ENDW | End a **WHILE** loop | endw |
| EQU | Define an assemble constant | <label> equ <expr> |
| ERROR | Issue an error message | error "<text_string>" |
| EXITM | Exit from a macro | exitm |
| EXPAND | Expand macro listing | expand |
| FOR | Perform counting loop **FOR** | for <iterator> = <expr1> to <expr2> [step <expr3>] |
| IF | Begin conditionally assembled code block | if <expr> |
| IFDEF | Execute if symbol has been defined | ifdef <label> |
| IFNDEF | Execute If symbol has not been defined | ifndef <label> |
| #INCLUDATA | Include binary data file | #includata "<data_file>" [,<address>] |
| #INCLUDE | Include additional source file | #include "<include_file>" |
| LINES | Re-declare line-per-page | lines <value> |
| LIST | Listing options | list [<list_option>,...,<list_option>] |
| LOCAL | Declare local macro variable | local <label>[,<label>] |
| MACRO | Declare macro definition | <label> macro [<arg>,...,<arg>] |
| MAXMACRODEPTH | Setup the maximum depth of macro expansion | Maxmacrodepth [=] <expr> |

| Directive | Description | Syntax |
|---|---|---|
| MESSG | Create user defined message | messg "<message_text>" |
| NEWPAGE | Re-declare line-per-page | Newpage <value> |
| NOEXPAND | Turn off macro expansion | noexpand |
| ORG | Set program origin | [<label>:] org <expr> |
| RADIX | Specify default radix | radix <default_radix> |
| REPEAT | Begin at-least-one-time loop | Repeat<br>  <statements><br>until <Boolean expression> |
| RES | Reserve memory block | [<label>:] res <mem_units> |
| SUBTITLE | Specify program subtitle | subtitle "<sub_text>" |
| SWITCH | Begin conditional switching assembly block | switch <expr> |
| TITLE | Specify program title | title "<title_text>" |
| #UNDEFINE | Delete a substitution label | #undefine <label> |
| UNTIL | End at-least-one-time loop if condition is true | Repeat<br>  <statements><br>until <Boolean expression> |
| VARIABLE | Declare symbol variable | variable <label>[=<expr>,...,<label>[=<expr>] ] |
| WHILE | Perform loop **WHILE** condition is true | while <expr> |

## 5.3 BREAK – Jump Out Point in a Logic Block

### 5.3.1 Syntax

***Syntax 1:***

```
<for|while|repeat – loop begin>
     [<statements>]
     break [<Boolean expr>]
     [<statements>]
<for|while|repeat – loop end>
```

***Syntax 2:***

```
switch <expr>
     case <expr1>[,<expr2>]
          [<statements>]
          break [<Boolean expr>]
          [<statements>]
     [<case-statements>]
endsw
```

### 5.3.2 Description

Set the logical point in a program which will escape the running flow from a WHILE, FOR, or REPEAT-UNIT loop. `break` also is used in switch block to achieve the purpose of switching among conditional branches.

### 5.3.3 Example

***Example 1:***

```
for i =0 to 4
nop
break i==2
halt
endfor
```

***Example 2:***

```
a=1
switch a
  case 1, 2
    nop
    break
case 1
    halt
endsw
```

### 5.3.4 See Also

FOR, WHILE, REPEAT, SWITCH

## 5.4 CASE – Define an Option Item of SWITCH

### 5.4.1 Syntax

```
switch <expr>
    case <expr1>[,<expr2>]
        [<statements>]
      :
      :
    default
        [<statements>]
endsw
```

### 5.4.2 Description

Define an option of selection statement. Once the <expr> matched one of the conditions after case, running flow will branch into that case item. case is part of a switch block, and must be used with switch

### 5.4.3 Example

```
a=1
switch a
  case 1, 2
     nop
     break
 case 1
     halt
endsw
```

### 5.4.4 See Also

DEFAULT, SWITCH

## 5.5 CBLOCK – Define a Block of Constants

### 5.5.1 Syntax

```
cblock [<expr>]
  [<label>[=<increment>][,<label>[=<increment>]]]
endc
```

### 5.5.2 Description

Define a list of named constants. Each <label> is assigned a value of one higher than the previous <label>. The purpose of this directive is to assign address offsets to many labels. The list of names end when an ENDC directive is encountered. <expr> indicates the starting value for the first name in the block. If no expression is found, the first name will receive a value one higher than the final name in the previous CBLOCK. If the first CBLOCK in the source file has no <expr>, assigned values start with zero. If <increment> is specified, then the next <label> is assigned the value of <increment> higher than the previous <label>. Multiple names may be given on a line, separated by commas. cblock is useful for defining constants in program and data memory.

### 5.5.3 Example

```
cblock 0x20     ; name_1 will be assigned 20
  name_1, name_2  ; name_2 is 21
```

```
  name_3 =0x30, name_4  ; name_4 is assigned 30,name_4 is assigned 31.
endc
```

### 5.5.4 See Also

ENDC

## 5.6 CONSTANT – Declare Symbol Constant

### 5.6.1 Syntax

```
constant <label>=<expr> [...,<label>=<expr>]
```

### 5.6.2 Description

Creates symbols for use in AMCA expressions. Constants may not be reset after having once been initialized, and the expression must be fully resolvable at the time of the assignment. This is the principal difference between symbols declared as CONSTANT and those declared as VARIABLE. Otherwise, constants and variables may be used interchangeably in expressions.

### 5.6.3 Example

```
variable RecLength=64                ; Set Default RecLength
constant BufLength=512               ; Init BufLength
             :                       ; RecLength may
             :                       ; be reset later
             :                       ; in RecLength=128
             :                       ;
constant MaxMem=RecLength+BufLength  ;CalcMaxMem
```

### 5.6.4 See Also

VARIABLE

## 5.7 CONTINUE – Ignore Statements Afterward and Start Next Loop

### 5.7.1 Syntax

```
<for|while|repeat – loop begin>
     [<statements>]
     continue [<Boolean expr>]
     [<statements>]
<for|while|repeat – loop end>
```

### 5.7.2 Description

Set a logical point in a program which will ignore statements after `continue` in `WHILE`, `FOR`, or `REPEAT-UNITL` looping block, and jump to the begin of looping block containing `continue` directive.

### 5.7.3 Example

```
for i =0 to 4
 nop
 continue i==2
 halt
endfor
```

### 5.7.4 See Also

`FOR, WHILE, REPEAT`

## 5.8 DATA – Create Numeric and Text Data

### 5.8.1 Syntax

```
[<label>] data <expr>,[,<expr>,...,<expr>]
[<label>] data "<text_string>"[,"<text_string>",...]
[<label>] data ‘<char>’[,‘<char>’,...]
```

### 5.8.2 Description

Initialize one or more words of program memory with data. The data may be in the form of constants or expressions. The data may also consist of ASCII character strings, `<text_string>`, enclosed in single quotes for one character or double quotes for strings. Single character items are placed into the low byte of the word. Each character in a strings occupies a word. When generating a binary file, this directive can also be used to declare initialized data values.

### 5.8.3 Example

```
data label_1+10 ; constants
data 1,2,label_2 ; constants
data "testing 1,2,3" ; text string
data ‘N’ ; single character
```

### 5.8.4 See Also

`DB, DN, DT, DW`

## 5.9 DB – Declare Data of One Byte

### 5.9.1 Syntax

```
[<label>] db <expr>[,<expr>,...,<expr>]
```

### 5.9.2 Description

Reserve program memory words with packed 8-bit values. Multiple expressions continue to fill bytes consecutively until the end of expressions. Should there be an odd number of expressions, the last byte will be zero.

### 5.9.3 Example

```
db 't', 0x0f, 'e', 0x0f, 's', 0x0f, 't'
```

### 5.9.4 See Also

DATA, DN, DT, DW

## 5.10 DEFAULT – Define an Unconditional Item of SWITCH

### 5.10.1 Syntax

```
switch <expr>
    case <expr1>[,<expr2>]
        [<statements>]
      :
      :
    default
        [<statements>]
endsw
```

### 5.10.2 Description

Define an unconditional item of selection statement. Once no condition after `case` items matched `<expr>`, running flow will go into `default` item. `default` is part of a `switch` block, and must be used with `switch`

### 5.10.3 Example

```
a=1
switch a
  case 1, 2
    nop
        break
```

```
        case 1
            halt
        default
      nop
    endsw
```

#### 5.10.4 See Also

CASE, SWITCH

### 5.11 #DEFINE – Define a Text Substitution Label

#### 5.11.1 Syntax

```
#define <name> [<string>]
```

#### 5.11.2 Description

This directive defines a text substitution string. Wherever <name> is encountered in the assembly code, <string> will be substituted. Using the directive with no <string> causes a definition of <name> to be noted internally and may be tested for using the IFDEF directive.

#### 5.11.3 Example

```
#define length 20
#define control 0x19, 7
:
:
srbr control ; set bit 7 in 0x19
```

#### 5.11.4 See Also

#UNDEFINE, IFDEF, IFNDEF

### 5.12 DN – Declare Data of One Nibble

#### 5.12.1 Syntax

```
[<label>:] dn <expr> [, <expr>, ..., <expr>]
```

#### 5.12.2 Description

Reserve program memory words with packed 8-bit values. Multiple expressions continue to fill nibbles consecutively until the end of expressions. Should there not be the number of expressions the multiple of 4, the last nibble, last two or last three nibbles will be zero.

### 5.12.3 Example

```
dn  0xf, 0x1, 0xf, 0x2, 0xf, 0x3, 0xf
```

### 5.12.4 See Also

DATA, DB, DT, DW

## 5.13 DT – Define Table

### 5.13.1 Syntax

```
[<label>] dt <expr> [, <expr>, ..., <expr>]
```

### 5.13.2 Description

Generates a series of DW instructions, one instruction for each <expr>. Each <expr> must be an word value. Each character in a string is stored in word length.

### 5.13.3 Example

```
dt "A Message", 0
dt FirstValue, SecondValue, EndOfValues
```

### 5.13.4 See Also

DATA, DB, DN, DW

## 5.14 DW – Declare Data of One Word

### 5.14.1 Syntax

```
[<label>:] dw <expr>[,<expr>,...,<expr>]
```

### 5.14.2 Description

Reserve program memory words for data, initializing that space to specific values. Values are stored into successive memory locations and the location counter is incremented by one. Expressions may be literal strings and are stored as described in the DATA directive.

### 5.14.3 Example

```
dw 39, "diagnostic 39", (d_list*2+d_offset)
dw diagbase-1
```

### 5.14.4 See Also

DATA, DB, DN, DT

## 5.15 ELSE – Begin Alternative Assembly Block to IF

### 5.15.1 Syntax

```
else
```

### 5.15.2 Description

Used in conjunction with an `IF` directive to provide an alternative path of assembly code should the `IF` evaluate to false. `ELSE` may be used inside a regular program block or macro.

### 5.15.3 Example

```
speed macro rate
if rate < 50
dw slow
else
dw fast
endif
endm
```

### 5.15.4 See Also

`ENDIF, IF`

## 5.16 END – End Program Block

### 5.16.1 Syntax

```
end
```

### 5.16.2 Description

Indicates the end of the program.

### 5.16.3 Example

```
list p=17c42
:    ; executable code
:    ;
end ; end of instructions
```

## 5.17 ENDC – End an Automatic Constant Block

### 5.17.1 Syntax

```
endc
```

### 5.17.2 Description

ENDC terminates the end of a CBLOCK list. It must be supplied to terminate the list.

### 5.17.3 See Also

CBLOCK

## 5.18 ENDFOR – End a For Loop

### 5.18.1 Syntax

```
endfor
```

### 5.18.2 Description

ENDFOR terminates a FOR loop. As long as the looping counter specified by the FOR directive went over the conditional boundary, the source code between the FOR directive and the ENDFOR directive will be repeatedly expanded in the assembly source code stream. This directive may be used inside a regular program block or macro.

### 5.18.3 See Also

FOR

## 5.19 ENDIF – End Conditional Assembly Block

### 5.19.1 Syntax

```
endif
```

### 5.19.2 Description

This directive marks the end of a conditional assembly block. ENDIF may be used inside a regular program block or macro.

### 5.19.3 See Also

ELSE, IF

## 5.20 ENDM – End a Macro Definition

### 5.20.1 Syntax

```
endm
```

### 5.20.2 Description

Terminates a macro definition begun with MACRO.

#### 5.20.3 Example

```
make_table macro arg1, arg2
  dw arg1, 0  ; null terminate table name
  res arg2    ; reserve storage
endm
```

#### 5.20.4 See Also

MACRO, EXITM

### 5.21 ENDS – Coding Convenience

#### 5.21.1 Syntax

```
ends
```

#### 5.21.2 Description

Present ENDFOR, ENDW, ENDSW, ENDIF

#### 5.21.3 See Also

ENDFOR, ENDW, ENDSW, ENDIF

### 5.22 ENDSW – End a Switch Block

#### 5.22.1 Syntax

```
endsw
```

#### 5.22.2 Description

Terminates a SWITCH block definition begun with SWITCH.

#### 5.22.3 Example

See the example for SWITCH

#### 5.22.4 See Also

SWITCH

### 5.23 ENDW – End a While Loop

#### 5.23.1 Syntax

```
endw
```

### 5.23.2 Description

ENDW terminates a WHILE loop. As long as the condition specified by the WHILE directive remains true, the source code between the WHILE directive and the ENDW directive will be repeatedly expanded in the assembly source code stream. This directive may be used inside a regular program block or macro.

### 5.23.3 Example

See the example for WHILE

### 5.23.4 See Also

WHILE

## 5.24 EQU – Define an Assembler Constant

### 5.24.1 Syntax

```
<label> equ <expr>
```

### 5.24.2 Description

The value of <expr> is assigned to <label>.

### 5.24.3 Example

```
four equ 4 ; assigned the numeric value of 4 to label four
```

## 5.25 ERROR – Issue an Error Message

### 5.25.1 Syntax

```
error "<text_string>"
```

### 5.25.2 Description

<text_string> is printed in a format identical to any AMCA error message. <text_string> may be from 1 to 80 characters.

### 5.25.3 Example

```
error_checking macro arg1
  if arg1 >= 55                ; if arg is out of range
  error "error_checking-01 arg out of range"
  endif
endm
```

#### 5.25.4 See Also

MESSG

### 5.26 EXITM – Exit from a Macro

#### 5.26.1 Syntax

```
exitm
```

#### 5.26.2 Description

Force immediate return from macro expansion during assembly. The effect is the same as if an ENDM directive had been encountered.

#### 5.26.3 Example

```
test macro filereg
    if filereg == 1 ; check for valid file
    exitm
    else
    error "bad file assignment"
    endif
endm
```

#### 5.27.4 See Also

ENDM MACRO

### 5.27 EXPAND – Expand Macro Listing

#### 5.27.1 Syntax

```
expand
```

#### 5.27.2 Description

Expand all macros in the listing file. This directive is roughly equivalent to the “Macro Expansion” assembler option, but may be disabled by the occurrence of a subsequent NOEXPAND.

#### 5.27.3 See Also

MACRO, NOEXPAND

### 5.28 FOR – Perform For Loop While Iterator Meets the Condition

#### 5.28.1 Syntax

```
for <iterator>=<expr1> to <expr2> [step <expr3>]
```

```
:
:
endfor
```

### 5.28.2 Description

The lines between the FOR and the ENDFOR are assembled as long as <iterator> evaluates in the range of <expr1> to <expr2>. A FOR loop can be repeated a maximum of 256 times.

### 5.28.3 Example

```
for I=0 to 5
   nop
endfor
```

### 5.28.4 See Also

ENDFOR

## 5.29 IF – Begin Conditionally Assembled Code Block

### 5.29.1 Syntax

```
if <expr>
```

### 5.29.2 Description

Begin execution of a conditional assembly block. If <expr> evaluates to true, the code immediately following the IF will assemble. Otherwise, subsequent code is skipped until an ELSE directive or an ENDIF directive is encountered. An expression that evaluates to zero is considered logically FALSE. An expression that evaluates to any other value is considered logically TRUE. The IF and WHILE directives operate on the logical value of an expression. A relational TRUE expression is guaranteed to return a nonzero value, FALSE a value of zero.

### 5.29.3 Example

```
if version == 100; check current version
  :      ;executable cod
  :      ;executable cod
else
   :      ;executable cod
  :      ;executable cod
endif
```

### 5.29.4 See Also

ELSE, ENDIF

## 5.30 IFDEF – Execute If Symbol has Been Defined

### 5.30.1 Syntax

```
ifdef <label>
```

### 5.30.2 Description

If <label> has been previously defined, usually by issuing a #DEFINE directive or by setting the value on the AMCA command line, the conditional path is taken. Assembly will continue until a matching ELSE or ENDIF directive is encountered.

### 5.30.3 Example

```
#define testing 1 ; set testing "on"
:
:
ifdef testing
  <execute test code> ; this path would be executed.
Endif
```

### 5.30.4 See Also

#DEFINE, ELSE, ENDIF, IFNDEF, #UNDEFINE

## 5.31 IFNDEF – Execute If Symbol has not Been Defined

### 5.31.1 Syntax

```
ifndef <label>
```

### 5.31.2 Description

If <label> has not been previously defined, or has been undefined by issuing an #UNDEFINE directive, then the code following the directive will be assembled. Assembly will be enabled or disabled until the next matching ELSE or ENDIF directive is encountered.

### 5.31.3 Example

```
#define testing1 ; set testing on
:
:
#undefine testing1 ; set testing off
```

```
ifndef testing ; if not in testing mode
: ; execute this path
:
endif
end ; end of source
```

#### 5.31.4 See Also

#DEFINE, ELSE, ENDIF, IFDEF, #UNDEFINE

## 5.32 #INCLUDATA – Include Binary Data File

### 5.32.1 Syntax

```
#includata "<binary_data_file>"[, address]
```

### 5.32.2 Description

The specified file is read in as binary data. The effect is the same as if the entire text of the included file were inserted into the file at the location of the #includata statement. If the includes data file needs to be inserted at a specific location, users can specify the location by `address`. `#includata` must be the last statement before `end` directive. `<binary_data_file>` must be enclosed in quotes. If a fully qualified path is specified, only that path will be searched. Otherwise, the search path is: source file directory. `<binary_data_file> will becomes a label after assembled.`

### 5.32.3 Example

```
#includata "c:\music\s02.sog", 0x2000 ; insert data file at 0x2000
#includata "s01" ; extension file name can be ignored
```

## 5.33 #INCLUDE – Include Additional Source File

### 5.33.1 Syntax

```
#include "<include_file>"
```

### 5.33.2 Description

The specified file is read in as source code. The effect is the same as if the entire text of the included file were inserted into the file at the location of the include statement. Upon end-of-file, source code assembly will resume from the original source file. `<include_file>` must be enclosed in quotes. If a fully qualified path is specified, only that path will be searched. Otherwise, the search path is: source file directory.

### 5.33.3 Example

```
#include "c:\sys\sysdefs.inc" ; system defs
#include "regs.h" ; register defs
```

## 5.34 LINES – Insert Listing Page Eject

### 5.34.1 Syntax

```
lines <value>
```

### 5.34.2 Description

Inserts a page eject into the listing file.

### 5.34.3 See Also

NEWPAGE

## 5.35 LIST – Listing Options

### 5.35.1 Syntax

```
list [<list_option>, ..., <list_option>]
```

### 5.35.2 Description

Occurring on a line by itself, the LIST directive has the effect of turning listing output on, if it had been previously turned off. Otherwise, one of the following list options can be supplied to control the assembly process or format the listing file:

**Table 4.2: List Directive Options**

| OPTION | DEFAULT | DESCRIPTION |
|---|---|---|
| c | On | Enable/Disable case sensitivity<br>c=on Enable<br>c=off Disable |
| d | None | Code number settings:<br>d=<3-digital number><br>where <3-digital number> will be the extended file name of O file and R file. |
| f | None | Configuration block file settings:<br>f=<cfgblk filename><br>where <cfgblk filename> is a file name string without path information and this cfgblk file must locates in the folder of source file |
| l | 19 | Source file lines per page settings:<br>l=<line-per-page><br>where <line-per-page> has default value 19; <line-per-page> >= 10 and <line-per-page> <= 1000 |

| OPTION | DEFAULT | DESCRIPTION |
|---|---|---|
| n | 256 | Maximum looping count settings:<br>n=<numeric value><br>where <numeric value> will prevent AMCA from the infinite directive looping of **WHILE** and **REPEAT** |
| p | None | Set the processor type:<br>/p=<processor_type><br>where <processor_type> is an ALPHA MCU device. For example, AM4EB005x |
| r | HEX | source file radix settings:<br>/r=<radix><br>where where `<radix>` is one of `[ hex \| dec \| oct \| bin ]` |
| unlockrsvmem | Locked | /unlockrsvmem<br>For 4-bit MCU only. Allow the programming right in reserved memory area |
| nocfgblk | Configuration Block required | /nocfgblk<br>For 4-bit MCU only. Ignore the assembly time check for the existence of configuration block file |

### 5.35.3 Example

```
list p=am4ea000x, c=off, r=dec
```

## 5.36 LOCAL – Declare Local Macro Variable

### 5.36.1 Syntax

```
local <label>[,<label>...]
```

### 5.36.2 Description

Declares that the specified data elements are to be considered in local context to the macro. `<label>` may be identical to another label declared outside the macro definition; there will be no conflict between the two. If the macro is called recursively, each invocation will have its own local copy.

### 5.36.3 Example

```
<main code segment>
:
:
len equ 10 ; global version
size equ 20 ; note that a local variable may now be created and modified
test macro size
    local len, label ; local len and label
    len set size ; modify local len
```

```
    label res len ; reserve buffer
    len set len-20 ;
endm ; end macro
```

### 5.36.4 See Also

ENDM, MACRO

## 5.37 MACRO – Declare Macro Definition

### 5.37.1 Syntax

```
<label> macro [<arg>, ..., <arg>]
```

### 5.37.2 Description

A macro is a sequence of instructions that can be inserted in the assembly source code by using a single macro call. The macro must first be defined, then it can be referred to in subsequent source code. A macro can call another macro, or may call itself recursively.

### 5.37.3 Example

```
Read macro device, buffer, count
Movia device
movar 0x10
movia buffer
movar 0x12
movia count
movar 0x13
endm
:
:
read 1,2,3
```

### 5.37.4 See Also

ELSE, ENDIF, ENDM, EXITM, IF, LOCAL

## 5.38 MAXMACRODEPTH – Define Maximum Macro Depth

### 5.38.1 Syntax

```
maxmacrodepth[=]<expr>
```

### 5.38.2 Description

MAXMACODEPTH defines the maximum valid macro depth to <expr>. <expr> must be less than or equal to the maximum depth 256. MAXMACODEPTH can be used more than once in a source file. Each use redefines the maximum valid macro depth.

### 5.38.3 Example

```
list p=am4ea000x
maxmacrodepth 0x10
:
:
```

## 5.39 MESSG – Create User Defined Message

### 5.39.1 Syntax

```
messg "<message_text>"
```

### 5.39.2 Description

Causes an informational message to be printed in the listing file. Issuing a MESSG directive does not set any error return codes.

### 5.39.3 Example

```
mssg_macro macro
    messg "mssg_macro-001 invoked without argument"

endm
```

### 5.39.4 See Also

ERROR

## 5.40 NEWPAGE – Insert Listing Page Eject

### 5.40.1 Syntax

```
newpage <value>
```

### 5.40.2 Description

Inserts a page eject into the listing file.

### 5.40.3 See Also

LINE

## 5.41 NOEXPAND – Turn off Macro Expansion

### 5.41.1 Syntax

```
noexpand
```

### 5.41.2 Description

Turns off macro expansion in the listing file.

### 5.41.3 See Also

EXPAND

## 5.42 ORG – Set Program Origin

### 5.42.1 Syntax

```
[<label>:] org <expr>
```

### 5.42.2 Description

Set the program origin for subsequent code at the address defined in <expr>. If <label> is specified, it will be given the value of the <expr>. If no ORG is specified, code generation will begin at address zero.

### 5.42.3 Example

```
int_1: org 0x20
; Vector 20 code goes here
int_2: org int_1+0x10
; Vector 30 code goes here
```

## 5.43 RADIX – Specify Default Radix

### 5.43.1 Syntax

```
radix <default_radix>
```

### 5.43.2 Description

Sets the default radix for data expressions. The default radix is hex. Valid radix values are: hex, dec, oct, or bin.

### 5.43.3 Example

```
radix dec
```

### 5.43.4 See Also

LIST

## 5.44 REPEAT – Begin a Repeat-Until Loop Block Definition

### 5.44.1 Syntax

```
repeat
:
:
until <expr>
```

### 5.44.2 Description

Begin a REPEAT-UNTIL block definition

### 5.44.3 Example

```
test_mac macro count
  variable i
  i = 0
  repeat
  i += 1
  until i > count
endm
:
:
end
```

### 5.44.4 See Also

WHILE, UNTIL

## 5.45 RES – Reserve Memory

### 5.45.1 Syntax

```
[<label>:] res <mem_units>
```

### 5.45.2 Description

Causes the memory location pointer to be advanced from its current location by the value specified in <mem_units>. <label> is assumed to be a program memory address. Address locations are defined in words.

### 5.45.3 Example

```
buffer: res 64 ; reserve 64 address locations of storage
```

### 5.45.4 See Also

ORG

## 5.46 SUBTITLE – Specify Program Subtitle

### 5.46.1 Syntax

```
subtitle "<sub_text>"
```

### 5.46.2 Description

<sub_text> is an ASCII string enclosed in double quotes, 60 characters or less in length. This directive establishes a second program header line for use as a subtitle in the listing output.

### 5.46.3 Example

```
subtitle "diagnostic section"
```

### 5.46.4 See Also

TITLE

## 5.47 SWITCH – Begin Conditional Switching Assembly Block

### 5.47.1 Syntax

```
switch <expr>
    case <expr1>[,<expr2>]
        [<statements>]
    case < exprM>[,<exprN>]
      :
      :
    default
        [<statements>]
endsw
```

### 5.47.2 Description

Begin execution of a conditional switching assembly block. If <expr> evaluates to matching any <exprX> after cases , the code immediately following that matched case will assemble. Otherwise, subsequent code is skipped until a default directive or an ENDSW directive is encountered.

### 5.47.3 Example

```
a=1
switch a
  case 1, 2
   nop
       break
   case 1
       halt
  default

endsw
```

### 5.47.4 See Also

BREAK, DEFAULT

## 5.48 TITLE – Specify Program Title

### 5.48.1 Syntax

```
title "<title_text>"
```

### 5.48.2 Description

<title_text> is a printable ASCII string enclosed in double quotes. It must be 60 characters or less. This directive establishes the text to be used in the top line of each page in the listing file.

### 5.48.3 Example

```
title "operational code, rev 5.0"
```

### 5.48.4 See Also

SUBTITLE

## 5.49 #UNDEFINE – Delete a Substitution Label

### 5.49.1 Syntax

```
#undefine <label>
```

### 5.49.2 Description

<label> is an identifier previously defined with the #DEFINE directive. It must be a valid AMCA label. The symbol named is removed from the symbol table.

### 5.49.3 Example

```
#define length 20
:
:
#undefine length
```

### 5.49.4 See Also

#DEFINE, IFDEF, #INCLUDE, IFNDEF

## 5.50 UNTIL – Perform Loop Until Condition is True

### 5.50.1 Syntax

```
repeat
:
:
until <expr>
```

### 5.50.2 Description

The lines between the REPEAT and the UNTIL are assembled at least once, and as long as <expr> evaluates to FALSE. A REPEAT loop can be repeated at maximum of 256 times.

### 5.50.3 Example

```
test_mac macro count
  variable i
  i = 0
  repeat
  i += 1
  until i < count
endm
:
:
end
```

### 5.50.4 See Also

WHILE, REPEAT

## 5.51 VARIABLE – Declare Symbol Variable

### 5.51.1 Syntax

```
variable <label>[=<expr>][,<label>[=<expr>]...]
```

### 5.51.2 Description

Creates symbols for use in AMCA expressions. Variables and constants may be used interchangeably in expressions. The VARIABLE directive creates a symbol that is functionally equivalent to those created by the SET directive. The difference is that the VARIABLE directive does not require that symbols be initialized when they are declared. Note that variable values cannot be updated within an operand. You must place variable assignments, increments, and decrements on separate lines.

### 5.51.3 Example

Please refer to the example given for the CONSTANT directive.

### 5.51.4 See Also

CONSTANT

## 5.52 WHILE – Perform Loop While Condition is True

### 5.52.1 Syntax

```
while <expr>
:
:
endw
```

### 5.52.2 Description

The lines between the WHILE and the ENDW are assembled as long as <expr> evaluates to TRUE. An expression that evaluates to zero is considered logically FALSE. An expression that evaluates to any other value is considered logically TRUE. A relational TRUE expression is guaranteed to return a non-zero value; FALSE a value of zero. A WHILE loop can contain at most 100 lines and be repeated a maximum of 256 times.

### 5.52.3 Example

```
test_mac macro count
variable i
i = 0
while i < count
```

```
movlw i
i += 1
endw
endm
start
test_mac 5
end
```

### 5.52.4 See Also

ENDW IF

# 6. Macro Language

Macros are user defined sets of instructions and directives that will be evaluated in-line with the assembler source code whenever the macro is invoked. Macros consist of sequences of assembler instructions and directives. They can be written to accept arguments, making them quite flexible. Their advantages are:

• Higher levels of abstraction, improving readability and reliability.

• Consistent solutions to frequently performed functions.

• Simplified changes.

• Improved testability.

Applications might include creating complex tables, frequently used code, and complex operations.

Topics covered in this chapter:

• Macro Syntax

• Macro Directives

• Text Substitution

• Macro Usage

## 6.1 Macro Syntax

AMCA macros are defined according to the following syntax:

```
<label> macro [<arg1>,<arg2> ..., <argn>]
:
:
endm
```

where `<label>` is a valid AMCA label and `<arg>` is any number of optional arguments supplied to the macro. The values assigned to these arguments at the time the macro is invoked will be substituted wherever the argument name occurs in the body of the macro. The body of a macro may be comprised of AMCA directives, or AMCA Macro Directives (`LOCAL` for example). Refer to Chapter 4. AMCA continues to process the body of the macro until an `EXITM` or `ENDM` directive is encountered.

**Note:** Forward references to macros are not permitted.

## 6.2 Macro Directives

There are directives that are unique to macro definitions. They cannot be used out of the macro context (refer to Chapter 7 for details concerning these directives):

• MACRO

• LOCAL

• EXITM

• ENDM

When writing macros, you can use any of these directives PLUS any other directives supported by AMCA.

## 6.3 Text Substitution

String replacement and expression evaluation may appear within the body of a macro. Arguments may be used anywhere within the body of the macro.

| Command | Description |
|---|---|
| <arg> | Substitute the argument text supplied as part of the macro invocation. |

```
define_table macro num_of_entry
    local a = 0
    while a < num_of_entry
    dw 0
    a += 1
    endw
endm
```

when invoked, would generate:

```
dw 0    ; 1st
dw 0    ; 2nd
:
:
dw 0    ; (num_of_entry-1)-th
dw 0    ; (num_of_entry)-th
```

## 6.4 Macro Usage

Once the macro has been defined, it can be invoked at any point within the source module by using a macro call, as described below:

```
<macro_name> [<arg>, ..., <arg>]
```

where <macro_name> is the name of a previously defined macro and arguments are supplied as required. The macro call itself will not occupy any locations in memory. However, the macro expansion will begin at the current memory location. Commas may be used to reserve an argument position. In this case, the argument will be an empty string. The argument list is terminated by white space or a semicolon. The EXITM directive (see Chapter 4) provides an alternate method for terminating a macro expansion. During a macro expansion, this directive causes expansion of the current macro to stop and all code between the EXITM and the ENDM directives for this macro to be ignored. If macros are nested, EXITM causes code generation to return to the previous level of macro expansion.

# 7. Expression Syntax and Operation

This chapter describes various expression formats, syntax, and operations used by AMCA.

Topics covered in this chapter:

- Text Strings
- Numeric Constants and Radix
- Arithmetic Operators and Precedence
- High/Mid/Low and Increment/Decrement Operators

## 7.1 Text Strings

A “string” is a sequence of any valid ASCII character (of the decimal range of 0 to 127) enclosed by double quotes. Strings may be of any length that will fit within a 255 column source line. If a matching quote mark is found, the string ends. If none is found before the end of the line, the string will end at the end of the line. While there is no direct provision for continuation onto a second line, it is generally no problem to use a second `DW` directive for the next line. The `DW` directive will store the entire string into successive words. If a string has an odd number of characters (bytes), the `DB` directive will pad the end of the string with one byte of zero (00).

See the examples below for the object code generated by different statements involving strings.

```
000010 0074 0065 0073 0-0007    dw "testing output string one\n"
       0074 0069 006E
       0067 0020 006F
       0075 0074 0070
       0075 0074 0020
       0073 0074 0072
       0069 006E 0067
       0020 006F 006E
       0065 005C 006E
                       0-0010    #define str "testing output string two"
000050 0074 0065 0073 0-0011    data "testing first output string"
       0074 0069 006E
       0067 0020 0066
       0069 0072 0073
       0074 0020 006F
       0075 0074 0070
       0075 0074 0020
       0073 0074 0072
       0069 006E 0067
```

```
000080 7465 7374 696E 0-0013    db "testing first output string"
       6720 6669 7273
       7420 6F75 7470
       7574 2073 7472
       696E 6700
```

## 7.2 Numeric Constants and Radix

AMCA supports the following radix forms: hexadecimal, decimal, octal, binary, and ASCII. The default radix is hexadecimal; the default radix determines what value will be assigned to constants in the object file when a radix is not explicitly specified by a base descriptor. AMCA only supports unsigned constants and the values are assumed to be positive.

**Note:** Intermediate values in constant expressions are treated as 32-bit unsigned integers. Whenever an attempt is made to place a constant in a field for which it is too large, a truncation warning will be issued.

The following table presents the various radix specifications:

**Table 7.1: Radix Specifications**

| Type | Syntax | Example |
|---|---|---|
| **Decimal** | D'<digits>' | D'100' |
| **Hexadecimal** | H'<hex_digits>'<br>0x<hex_digits><br><hex_digits>h | H'9f'<br>0x9f<br>9fh |
| **Octal** | O'<octal_digits>' | O'777' |
| **Binary** | B'<binary_digits>' | B'00111001' |
| **ASCII** | '<character>'<br>A'<character>' | 'C'<br>A'C' |

**Table 6.2: Arithmetic Operators and Precedence**

| Operator | | Example |
|---|---|---|
| $ | Current/Return program counter | goto $ + 3 |
| ( | Left Parenthesis | 1 + (d * 4) |
| ) | Right Parenthesis | (Length + 1) * 256 |
| ! | Item NOT (logical complement) | if ! (a == b) |
| - | Negation (2's complement) | -1 * Length |
| ~ | Complement | flags = ~flags |
| high | Return high byte of a 24-bit value | movia high 0x121314<br>;accumulator will contain 0x12 |
| mid | Return mid byte of a 24-bit value | movia mid 0x121314<br>;accumulator will contain 0x13 |
| low | Return low byte of a 24-bit value | movia low 0x121314<br>;accumulator will contain 0x14 |

| Operator | | Example |
|---|---|---|
| high0 | Return low nibble of high byte of a 24-bit value | movia high0 0x123456<br>;accumulator will contain 0x2 |
| high1 | Return high nibble of high byte of a 24-bit value | movia high1 0x123456<br>;accumulator will contain 0x1 |
| mid0 | Return low nibble of middle byte of a 24-bit value | movia mid0 0x123456<br>;accumulator will contain 0x4 |
| mid1 | Return high nibble of middle byte of a 24-bit value | movia mid1 0x123456<br>;accumulator will contain 0x3 |
| low0 | Return low nibble of low byte of a 24-bit value | movia low0 0x123456<br>;accumulator will contain 0x6 |
| low1 | Return high nibble of low byte of a 24-bit value | movia low1 0x123456<br>;accumulator will contain 0x5 |
| * | Multiply | a = b * c |
| / | Divide | a = b / c |
| % | Modulus | entry_len = tot_len % 16 |
| + | Add | tot_len = entry_len * 8 + 1 |
| - | Subtract | entry_len = (tot - 1) / 8 |
| << | Left shift | flags = flags << 1 |
| >> | Right shift | flags = flags >> 1 |
| >= | Greater or equal | if entry_idx >= num_entries |
| > | Greater than | if entry_idx > num_entries |
| < | Less than | if entry_idx < num_entries |
| <= | Less or equal | if entry_idx <= num_entries |
| == | Equal to | if entry_idx == num_entries |
| != | Not equal to | if entry_idx != num_entries |
| & | Bitwise AND | flags = flags & ERROR_BIT |
| ^ | Bitwise exclusive OR | flags = flags ^ ERROR_BIT |
| \| | Bitwise inclusive OR | flags = flags \| ERROR_BIT |
| && | Logical AND | if (len == 512) && (b == c) |
| \|\| | Logical OR | if (len == 512) \|\| (b == c) |
| = | Set equal to | entry_index = 0 |
| += | Add to, set equal | entry_index += 1 |
| -= | Subtract, set equal | entry_index -= 1 |
| *= | Multiply, set equal | entry_index *= entry_length |
| /= | Divide, set equal | entry_total /= entry_length |
| %= | Modulus, set equal | entry_index %= 8 |
| <<= | Left shift, set equal | flags <<= 3 |

| Operator | | Example |
|---|---|---|
| >>= | Right shift, set equal | `flags >>= 3` |
| &= | AND, set equal | `flags &= ERROR_FLAG` |
| \|= | Inclusive OR, set equal | `flags \|= ERROR_FLAG` |
| ^= | Exclusive OR, set equal | `flags ^= ERROR_FLAG` |
| ++ | Increment | `i ++` |
| -- | Decrement | `i --` |

## 7.3 High/Mid/Low

### 7.3.1 Syntax

```
high <operand>
mid <operand>
low <operand>
```

### 7.3.2 Description

These operators are used to return one byte of a multi-byte label value. This is done to handle dynamic pointer calculations as might be used with table read and write instructions.

## 7.4 Increment/Decrement (++/--)

### 7.4.1 Syntax

```
<variable>++
<variable>--
```

### 7.4.2 Description

Increments or decrements a variable value. These operators can only be used on a line by themselves; they cannot be embedded within other expression evaluation.

### 7.4.3 Example

```
LoopCount = 4
while LoopCount > 0
  nop
  LoopCount --
Endw
```

# Appendix A. Quick Reference

This appendix lists abbreviated information on AMCA and MCU instruction sets for use in developing applications using AMCA.

Topics covered in this appendix:

• AMCA Quick Reference

• 4-Bit MCU List

• 8-Bit MCU List

• 4-Bit MCU Instruction Set

## A.1 AMCA Quick Reference

The following Quick Reference Guide gives all the instructions, directives, and command list options for AMCA Assembler.

**Table A.1: AMCA Directive Language Summary**

| Directive | Description | Syntax |
|---|---|---|
| **CONTROL DIRECTIVES** | | |
| CONSTANT | Declare symbol constant | constant <label>[=<expr>,...,<label>[=<expr>] ] |
| #DEFINE | Define a text substitution label | #define <name> [<value>]<br>#define <name> [<arg>,...,<arg>] |
| END | End program block | end |
| EQU | Define an assemble constant | <label> equ <expr> |
| ERROR | Issue an error message | error "<text_string>" |
| #INCLUDATA | Include binary data file | #includata "<data_file>" [,<address>] |
| #INCLUDE | Include additional source file | #include "<include_file>" |
| LIST | Listing options | list [<list_option>,...,<list_option>] |
| MESSG | Create user defined message | messg "<message_text>" |
| ORG | Set program origin | [<label>:] org <expr> |
| LINES | Re-declare line-per-page | lines <value> |
| NEWPAGE | Re-declare line-per-page | Newpage <value> |
| RADIX | Specify default radix | radix <default_radix> |
| SUBTITLE | Specify program subtitle | subtitle "<sub_text>" |
| TITLE | Specify program title | title "<title_text>" |
| #UNDEFINE | Delete a substitution label | #undefine <label> |
| VARIABLE | Declare symbol variable | variable <label>[=<expr>,...,<label>[=<expr>] ] |
| **CONDITIONAL ASSEMBLY** | | |
| BREAK | Escape from a **FOR**, **WHILE** or **REPEAT-UNTIL** loop, or Jump to the end of a **SWITCH** block | break [<Boolean expression>] |

| Directive | Description | Syntax |
|---|---|---|
| CASE | Part of a **SWITCH** block; must use **CASE** with **SWITCH** | switch <expression><br>case <expression 1>[,<expression 2>]<br><statements> |
| CONTINUE | Jump to the begin of **FOR**, **WHILE** or **REPEAT-UNTIL** loop that contains **CONTINUE** directive.<br><br>All statements behind **CONTINUE** in a loop are ignored. | continue [<Boolean expression>] |
| DEFAULT | Part of a **SWITCH** block; must use **DEFAULT** with **SWITCH.**<br>Begin default assembly block to **SWITCH** | default<br><statements> |
| ELSE | Begin alternative assembly block to **IF** | else<br><statements> |
| ENDFOR | End a **FOR** loop | endfor |
| ENDIF | End conditional assembly block | endif |
| ENDS | Directive for coding convenience: presenting **ENDFOR, ENDW, ENDSW, ENDIF** | ends |
| ENDSW | End conditional switching assembly block | endsw |
| ENDW | End a **WHILE** loop | endw |
| FOR | Perform counting loop **FOR** | for <iterator> = <expr1> to <expr2> [step <expr3>] |
| IF | Begin conditionally assembled code block | if <expr> |
| IFDEF | Execute if symbol has been defined | ifdef <label> |
| IFNDEF | Execute If symbol has not been defined | ifndef <label> |
| REPEAT | Begin at-least-one-time loop | Repeat<br><statements><br>until <Boolean expression> |
| SWITCH | Begin conditional switching assembly block | switch <expr> |
| UNTIL | End at-least-one-time loop if condition is true | Repeat<br><statements><br>until <Boolean expression> |
| WHILE | Perform loop **WHILE** condition is true | while <expr> |
| **DATA** | | |
| CBLOCK | Define a block of constants | cblock [<expr>] |
| DATA | Create numeric and text data | [<label>] data <expr>,[,<expr>,...,<expr>]<br>[<label>] data "<text_string>"[,"<text_string>",...] |
| DB | Declare data of one byte | [<label>] db <expr>[,<expr>,...,<expr>]<br>[<label>] db "<text_string>"[,"<text_string>",...] |
| DN | Declare data of one nibble | [<label>] dn <expr>,[,<expr>,...,<expr>] |
| DT | Define table | [<label>] dt <expr>[,<expr>,...,<expr>]<br>[<label>] dt "<text_string>"[,"<text_string>",...] |
| DW | Declare data of one word | [<label>] dw <expr>[,<expr>,...,<expr>]<br>[<label>] dw "<text_string>"[,"<text_string>",...] |
| ENDC | End an automatic constant block | endc |
| RES | Reserve memory block | [<label>:] res <mem_units> |

| Directive | Description | Syntax |
|---|---|---|
| **MACRO** | | |
| ENDM | End a macro definition | endm |
| EXITM | Exit from a macro | exitm |
| EXPAND | Expand macro listing | expand |
| LOCAL | Declare local macro variable | local <label>[,<label>] |
| MACRO | Declare macro definition | <label> macro [<arg>,...,<arg>] |
| MAXMACRODEPTH | Setup the maximum depth of macro expansion | Maxmacrodepth [=] <expr> |
| NOEXPAND | Turn off macro expansion | noexpand |

**Table A.2: AMCA Assembler Options:**

| OPTION | DEFAULT | DESCRIPTION |
|---|---|---|
| c | On | Enable/Disable case sensitivity<br>c=on Enable<br>c=off Disable |
| d | None | Code number settings:<br>d=<3-digital number><br>where <3-digital number> will be the extended file name of O file and R file. |
| f | None | Configuration block file settings:<br>f=<cfgblk filename><br>where <cfgblk filename> is a file name string without path information and this cfgblk file must locates in the folder of source file |
| l | 19 | Source file lines per page settings:<br>l=<line-per-page><br>where <line-per-page> has default value 19; <line-per-page> >= 10 and <line-per-page> <= 1000 |
| n | 256 | Maximum looping count settings:<br>n=<numeric value><br>where <numeric value> will prevent AMCA from the infinite directive looping of **WHILE** and **REPEAT** |
| p | None | Set the processor type:<br>/p=<processor_type><br>where <processor_type> is an ALPHA MCU device. For example, AM4EB005x |
| r | HEX | source file radix settings:<br>/r=<radix><br>where where `<radix>` is one of `[ hex \| dec \| oct \| bin ]` |
| unlockrsvmem | Locked | /unlockrsvmem<br>For 4-bit MCU only. Allow the programming right in reserved memory area |
| nocfgblk | Configuration block required | /nocfgblk<br>For 4-bit MCU only. Ignore the assembly time check for the existence of configuration block file |

**Table A.3: Radix Types Supported**

| Type | Syntax | Example |
|---|---|---|
| **Decimal** | D'<digits>' | D'100' |
| **Hexadecimal** | H'<hex_digits>'<br>0x<hex_digits><br><hex_digits>h | H'9f'<br>0x9f<br>9fh |
| **Octal** | O'<octal_digits>' | O'777' |
| **Binary** | B'<binary_digits>' | B'00111001' |
| **ASCII** | '<character>'<br>A'<character>' | 'C'<br>A'C' |

**Table A.4: AMCA Arithmetic Operators**

| Operator | | Example |
|---|---|---|
| $ | Current/Return program counter | goto $ + 3 |
| ( | Left Parenthesis | 1 + (d * 4) |
| ) | Right Parenthesis | (Length + 1) * 256 |
| ! | Item NOT (logical complement) | if ! (a == b) |
| - | Negation (2's complement) | -1 * Length |
| ~ | Complement | flags = ~flags |
| high | Return high byte of a 24-bit value | movia high 0x121314<br>;accumulator will contain 0x12 |
| mid | Return mid byte of a 24-bit value | movia mid 0x121314<br>;accumulator will contain 0x13 |
| low | Return low byte of a 24-bit value | movia low 0x121314<br>;accumulator will contain 0x14 |
| high0 | Return low nibble of high byte of a 24-bit value | movia high0 0x123456<br>;accumulator will contain 0x2 |
| high1 | Return high nibble of high byte of a 24-bit value | movia high1 0x123456<br>;accumulator will contain 0x1 |
| mid0 | Return low nibble of middle byte of a 24-bit value | movia mid0 0x123456<br>;accumulator will contain 0x4 |
| mid1 | Return high nibble of middle byte of a 24-bit value | movia mid1 0x123456<br>;accumulator will contain 0x3 |
| low0 | Return low nibble of low byte of a 24-bit value | movia low0 0x123456<br>;accumulator will contain 0x6 |
| low1 | Return high nibble of low byte of a 24-bit value | movia low1 0x123456<br>;accumulator will contain 0x5 |
| * | Multiply | a = b * c |
| / | Divide | a = b / c |
| % | Modulus | entry_len = tot_len % 16 |
| + | Add | tot_len = entry_len * 8 + 1 |
| - | Subtract | entry_len = (tot - 1) / 8 |

| Operator | | Example |
|---|---|---|
| << | Left shift | `flags = flags << 1` |
| >> | Right shift | `flags = flags >> 1` |
| >= | Greater or equal | `if entry_idx >= num_entries` |
| > | Greater than | `if entry_idx > num_entries` |
| < | Less than | `if entry_idx < num_entries` |
| <= | Less or equal | `if entry_idx <= num_entries` |
| == | Equal to | `if entry_idx == num_entries` |
| != | Not equal to | `if entry_idx != num_entries` |
| & | Bitwise AND | `flags = flags & ERROR_BIT` |
| ^ | Bitwise exclusive OR | `flags = flags ^ ERROR_BIT` |
| \| | Bitwise inclusive OR | `flags = flags \| ERROR_BIT` |
| && | Logical AND | `if (len == 512) && (b == c)` |
| \|\| | Logical OR | `if (len == 512) \|\| (b == c)` |
| = | Set equal to | `entry_index = 0` |
| += | Add to, set equal | `entry_index += 1` |
| -= | Subtract, set equal | `entry_index -= 1` |
| *= | Multiply, set equal | `entry_index *= entry_length` |
| /= | Divide, set equal | `entry_total /= entry_length` |
| %= | Modulus, set equal | `entry_index %= 8` |
| <<= | Left shift, set equal | `flags <<= 3` |
| >>= | Right shift, set equal | `flags >>= 3` |
| &= | AND, set equal | `flags &= ERROR_FLAG` |
| \|= | Inclusive OR, set equal | `flags \|= ERROR_FLAG` |
| ^= | Exclusive OR, set equal | `flags ^= ERROR_FLAG` |
| ++ | Increment | `i ++` |
| -- | Decrement | `i --` |

## A.2 4-Bit MCU List

**Table A.5: 4-Bit MCU List**

| | IC type | PROG ROM size | DATA ROM size | Reserved Memory | I/O Pin Count | Comment |
|---|---|---|---|---|---|---|
| 1 | AM4DA000x | 4K | 1M | 0x0080—0x00C0 | 12 I/O | EV chip, not for product |
| 2 | AM4DB004x | 2K | 16K | 0x0080—0x03FF | 2 I/O | |
| 3 | AM4DC004x | 4K | 16K | 0x0080—0x03FF | 4 I/O, 2 O | |
| 4 | AM4DC009x | 4K | 32K | 0x0080—0x03FF | 4 I/O, 2 O | |
| 5 | AM4DC015x | 4K | 48K | 0x0080—0x03FF | 4 I/O, 2 O | |
| 6 | AM4DC020x | 4K | 64K | 0x0080—0x03FF | 4 I/O, 2 O | |
| 7 | AM4DC025x | 4K | 80K | 0x0080—0x03FF | 4 I/O, 2 O | |
| 8 | AM4DC031x | 4K | 96K | 0x0080—0x03FF | 4 I/O, 2 O | |
| 9 | AM4DC041x | 4K | 128K | 0x0080—0x03FF | 4 I/O, 2 O | |
| 10 | AM4DC052x | 4K | 160K | 0x0080—0x03FF | 4 I/O, 2 O | |
| 11 | AM4DC063x | 4K | 192K | 0x0080—0x03FF | 4 I/O, 2 O | |
| 12 | AM4DC073x | 4K | 224K | 0x0080—0x03FF | 4 I/O, 2 O | |
| 13 | AM4DC084x | 4K | 256K | 0x0080—0x03FF | 4 I/O, 2 O | |
| 14 | AM4DD009x | 4K | 32K | 0x0080—0x03FF | 4 I/O, 4 I, 4 O | |
| 15 | AM4DD020x | 4K | 64K | 0x0080—0x03FF | 4 I/O, 4 I, 4 O | |
| 16 | AM4DD031x | 4K | 96K | 0x0080—0x03FF | 4 I/O, 4 I, 4 O | |
| 17 | AM4DD041x | 4K | 128K | 0x0080—0x03FF | 4 I/O, 4 I, 4 O | |
| 18 | AM4DD063x | 4K | 192K | 0x0080—0x03FF | 4 I/O, 4 I, 4 O | |
| 19 | AM4DD084x | 4K | 256K | 0x0080—0x03FF | 4 I/O, 4 I, 4 O | |
| 20 | AM4DD127x | 4K | 384K | 0x0080—0x03FF | 4 I/O, 4 I, 4 O | |
| 21 | AM4DD170x | 4K | 512K | 0x0080—0x03FF | 4 I/O, 4 I, 4 O | |
| 22 | AM4DD255x | 4K | 768K | 0x0080—0x03FF | 4 I/O, 4 I, 4 O | |
| 23 | AM4DD340x | 4K | 1M | 0x0080—0x03FF | 4 I/O, 4 I, 4 O | |
| 24 | AM4EA000x | 64K | 4M | 4M | 24 I/O | EV chip, not for product |
| 25 | AM4EB005x | 8K | 16K | 16K | 4 I/O | |
| 26 | AM4EB010x | 8K | 32K | 32K | 4 I/O | |
| 27 | AM4EC005x | 16K | 16K | 16K | 8 I/O | |
| 28 | AM4EC010x | 16K | 32K | 32K | 8 I/O | |
| 29 | AM4EC015x | 16K | 48K | 48K | 8 I/O | |
| 30 | AM4EC020x | 16K | 64K | 64K | 8 I/O | |
| 31 | AM4EC031x | 16K | 96K | 96K | 8 I/O | |
| 32 | AM4EC041x | 16K | 128K | 128K | 8 I/O | |
| 33 | AM4EC052x | 16K | 160K | 160K | 8 I/O | |
| 34 | AM4EC063x | 16K | 192K | 192K | 8 I/O | |
| 35 | AM4ED005x | 16K | 16K | 16K | 12 I/O | |
| 36 | AM4ED006x | 16K | 20K | 20K | 12 I/O | |
| 37 | AM4ED010x | 32K | 32K | 32K | 12 I/O | |
| 38 | AM4ED011x | 16K | 32K | 32K | 12 I/O | |
| 39 | AM4ED015x | 16K | 48K | 48K | 12 I/O | |

| | IC type | PROG ROM size | DATA ROM size | Reserved Memory | I/O Pin Count | Comment |
|---|---|---|---|---|---|---|
| 40 | AM4ED020x | 16K | 64K | 64K | 12 I/O | |
| 41 | AM4ED031x | 16K | 96K | 96K | 12 I/O | |
| 42 | AM4ED041x | 16K | 128K | 128K | 12 I/O | |
| 43 | AM4ED052x | 16K | 160K | 160K | 12 I/O | |
| 44 | AM4ED063x | 16K | 192K | 192K | 12 I/O | |
| 45 | AM4ED073x | 16K | 224K | 224K | 12 I/O | |
| 46 | AM4ED084x | 16K | 256K | 256K | 12 I/O | |
| 47 | AM4EE010x | 16K | 32K | 32K | 16 I/O | |
| 48 | AM4EE015x | 16K | 48K | 48K | 16 I/O | |
| 49 | AM4EE020x | 16K | 64K | 64K | 16 I/O | |
| 50 | AM4EE031x | 16K | 96K | 96K | 16 I/O | |
| 51 | AM4EE041x | 16K | 128K | 128K | 16 I/O | |
| 52 | AM4EE063x | 16K | 192K | 192K | 16 I/O | |
| 53 | AM4EE084x | 64K | 256K | 256K | 16 I/O | |
| 54 | AM4EE106x | 64K | 320K | 320K | 16 I/O | |
| 55 | AM4EE127x | 64K | 384K | 384K | 16 I/O | |
| 56 | AM4EG127x | 64K | 384K | 384K | 24 I/O | |
| 57 | AM4EG170x | 64K | 512K | 512K | 24 I/O | |
| 58 | AM4EG255x | 64K | 768K | 768K | 24 I/O | |
| 59 | AM4EG340x | 64K | 1M | 1M | 24 I/O | |

## A.3 8-Bit MCU List

**Table A.6: 8-Bit MCU List**

| | IC type | PROG ROM size | DATA ROM size | Reserved Memory | I/O Pin Count | Comment |
|---|---|---|---|---|---|---|
| 1 | AM8AB008x | 64K | 512K | N/A | 33 I/O | |
| 2 | AM8AD000x | 64K | 16M | N/A | 33 I/O | EV chip, not for product |
| 3 | AM8AG001x | 64K | 64K | N/A | 33 I/O | |
| 4 | AM8AG002x | 64K | 128K | N/A | 33 I/O | |
| 5 | AM8AG004x | 64K | 256K | N/A | 33 I/O | |

## A.4 4-Bit MCU Instruction Set

**Table A.7: 4-Bit MCU Instruction Set**

| OPCode | | | | | | | | | | Mnemonic | Description | Function |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | | | |
| 0 | 1 | 0 | 0 | f | f | f | f | f | f | MOVAF | Move the data (6'bffffff) in ACC* to F** | F → ACC |
| 0 | 1 | 0 | 1 | f | f | f | f | f | f | MOVFA | Move the data (6'bffffff) in F to ACC | ACC→ F |
| 1 | 0 | 0 | r | r | r | r | r | r | r | MOVAR | Move the dada (7'brrrrrrr) in ACC to R***. | ACC → R |
| 1 | 0 | 1 | r | r | r | r | r | r | r | MOVRA | Move the dada (7'brrrrrrr) in R to ACC. | R → ACC |
| 1 | 1 | 0 | 0 | 0 | 0 | i | i | i | i | MOVIA | Move the immediate value I (4'biiii) to ACC | I→ ACC |
| 0 | 1 | 1 | 0 | 0 | r | r | r | r | r | ANDARA | A logic operation, AND, between the data in ACC and the data in R. The result of operation will be restored to A.CC | A CC• R → ACC |
| 0 | 1 | 1 | 0 | 1 | r | r | r | r | r | ORARA | A logic operation, OR, between the data in ACC and the data in R. The result of operation will be restored to ACC. | ACC + R → ACC |
| 0 | 1 | 1 | 1 | 0 | r | r | r | r | r | XORARA | A logic operation, XOR, between the data in ACC and the data in R. The result of operation will be restored to ACC after operating. | ACC ⊕ R → ACC |
| 1 | 1 | 0 | 0 | 0 | 1 | i | i | i | i | ANDIA | A logic operation, AND, between the immediate value and the data in ACC. The result of operation will be restored to ACC after operating. | I • ACC→ A |
| 1 | 1 | 0 | 0 | 1 | 0 | i | i | i | i | ORIA | A logic operation, OR, between the immediate value and the data in ACC. The result of operation will be restored to ACC after operating. | I + ACC→ ACC |
| 1 | 1 | 0 | 0 | 1 | 1 | i | i | i | i | XORIA | A logic operation, XOR, between the immediate value and the data in ACC. The result of operation will be restored to ACC after operating. | I ⊕ ACC → ACC |
| 0 | 1 | 1 | 1 | 1 | r | r | r | r | r | ADDARCA | A add operation for the data in ACC, the data in R and carry register. The result of operation will be restored to ACC after operating. | ACC + R + C→ ACC |
| 1 | 1 | 0 | 1 | 0 | r | r | r | r | r | INCRR | An increasing operation for the data in R and the result of operation will be restored to R after operating. | R + 1→ R |
| 1 | 1 | 0 | 1 | 1 | r | r | r | r | r | DECRR | A decreasing operation for the data in R and the result of operation will be restored to R after operating. | R - 1→ R |
| 1 | 1 | 1 | 0 | r | r | r | r | r | r | SREQA | A branch condition operation between the relation of the data in R and ACC. | ·Skip next if R=ACC<br>·Run next instruction if R ≠ ACC |
| 1 | 1 | 1 | 1 | 0 | r | r | r | r | r | SREQZ | A branch condition operation to judge that the data in R if equal zero. | ·Skip next if R=0<br>·Run next instruction if R ≠ 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | r | r | r | r | SAEQI | A branch condition operation to judge that the data in R if equal the immediate value. | ·Skip next if ACC=I<br>·Run next instruction if ACC ≠ I |
| 0 | 0 | x | x | x | x | x | x | x | x | JMP | Unconditional jump of program counter (PC). | X[7:0] → PC[7:0] |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | HALT | Enter sleep mode | Stop all operation in chip |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | PUSH | Push PC to Stack | PC +2 → STK |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | RET | Return from subroutine without pop PCH0 & PCH1 back. | STK → PC |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | IRET | Interrupt return from interrupt subroutine with pop PCH0 & PCH1 back. | {PCH1, PCH0, STK}→ PC |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | VDRDI | Read ROM data to voice latch (22H, 23H, 24H) and increase VPR. | ·ROM[3:0]→ VLH0<br>·ROM[7:4]→ VLH1<br>·ROM[9:8]→ VLH2<br>·VPR=VPR+1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | DTRDI | Read ROM[9:4] to Data Latch (20H, 21H), ROM[3:0] to ACC and increase DPR. | ·ROM[3:0]→ ACC<br>·ROM[7:4]→ TBLH1<br>·ROM[9:8]→ TBLH2<br>·DPR=DPR+1 |

| OPCode | | | | | | | | | | Mnemonic | Description | Function |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| b9 | b8 | b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 | | | |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | DTRD | Read ROM[9:4] to Data Latch (20H, 21H), ROM[3:0] to ACC. | ·ROM[3:0]→ ACC<br>·ROM[7:4]→ TBLH1<br>·ROM[9:8]→ TBLH2 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | MDRDI | Read ROM[9:4] to Data Latch (20H, 21H), ROM[3:0] to ACC and increase DPR. Then according to the channel bit to write TONE & OCT registers. | ·ROM[3:0]→ ACC<br>·ROM[7:4]→ TBLH1<br>·ROM[9:8]→ TBLH2<br>·When ROM[0]=1, OCT=ROM[3:1] and TONE=ROM[7:4].<br>· DPR=DPR+1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | NOP | It is a mnemonic of no operation. | PC=PC+1 |

*: It represents that it is accumulator.

**: It represents that it is function register.

***: It represents that it is general register (ram).

# Appendix B. AMCA Errors/Warnings

The following messages are produced by AMCA. These messages always appear in the listing file directly above each line in which the error occurred. The messages are stored in the error file (.ERR).

Topics covered in this appendix:

• Errors

• Warnings

## B.1 Errors

E0000 Encounter a recursive definition link

E0001 Defined symbol not found

E0002 Instruction redefinition is forbidden

E0003 Directive redefinition is forbidden

E0004 Cannot redefine symbol

E0005 Is a reserved word that cannot be redefined

E0006 Included file string not found

E0007 Cannot include multiple files

E0008 Invalid including string

E0009 Cannot open file

E000A Try to undefine a null string

E000B Ttoo many undefined symbols

E000C Undefined symbol not found

E000D Encounter redundant tokens

E000E Try to use undefined symbol

E000F Invalid symbol name

E0010 Try to modify a constant variable

E0011 User defined symbol cannot equal instruction name

E0012 User defined symbol cannot equal directive name

E0013 User defined symbol cannot equal reserved word

E0014 Divided by zero

E0015 Mod by zero

E0016 'endm' must be associated with 'macro'

E0017 Behind 'endm',no token is in need

E0018 Invalid hexadecimal number

E0019 Out of maximum value of hexadecimal number

E001A Radix setting is unrecognized

E001B Invalid octal number

E001C Out of maximum value of octal number

E001D Invalid decimal number
E001E Out of maximum value of decimal number
E001F Invalid binary number
E0020 Out of maximum value of binary number
E0021 Missing the paired quote
E0022 Invalid numeric value
E0023 Encounter an unknown operator
E0024 Need an operand
E0025 Missing the left-side parenthesis '('
E0026 Missing the left-side parenthesis '('
E0027 Unknown identifier
E0028 Cannot use unary operator
E0029 Operator is expected
E002A Encounter an unexpected operand
E002B Missing the right-side parenthesis ')'
E002C Missing the operator
E002D Unary operator is not allowed here
E002E The left-side parenthesis '(' is required
E002F User defined error message
E0030 Included data file is required
E0031 Included data file not found
E0032 Continue' should be in the directive loop
E0033 Case' should follow 'switch'
E0034 Missing the 'case' value
E0035 Case' exrpession not found
E0036 Constant not found
E0037 Unexpected ','
E0038 Identifier required
E0039 Invalid identifier name
E003A Redefined constant is aborted
E003B '=' is required before
E003C Initial expression expected
E003D ‘break' should be in the directive loop or 'switch'
E003E Empty 'db'
E003F Missing initial expression
E0040 'default' should be in the 'switch'-'endsw' block
E0041 Useless token(s) encountered

E0042 Empty 'dw'
E0043 Initial expression not found
E0044 'else' must stand along
E0045 'else' misplaced
E0046 'end' doesn't need any other token
E0047 'endc' doesn't need any other token
E0048 'endc' must follow the 'cblock'
E0049 'endfor' must be paired with 'for'
E004A 'endfor' should follow 'for'
E004B 'endif' should follow 'if','ifdef',or 'ifndef'
E004C 'endif' must stand along
E004D 'endif' misplaced
E004E 'endsw' misplaced
E004F 'endsw' should follow 'switch'
E0050 'ends' misplaced
E0051 'endw' should follow 'while'
E0052 'endw' misplaced
E0053 'endw' must stand along
E0054 Missing the initial value of 'equ' identifier
E0055 Invalid name of 'equ' identifier
E0056 'exitm' out of the macro definition
E0057 'for' clause is not completed
E0058 Unrecognized iterator
E0059 Stop-value expression not found
E005A Stop-expression not found
E005B Error occurs when initial value evaluated
E005C Stop expression not found
E005D Error occurs when stop value evaluated
E005E Step expression not found
E005F Error occurs when step value evaluated
E0060 Infinite loop is prohibited
E0061 Boolean expression expected
E0062 Depended symbol not found
E0063 Too many symbols encountered
E0064 Variable not found
E0065 Is not allowed here,token ',' is expected
E0066 L-value(identifier) expected

E0067 Radix setting not found
E0068 Unrecognized radix setting
E0069 'repeat' must stand along
E006A 'res' needs a setting expression
E006B Try to reserve a negative space
E006C Expression required
E006D Associated 'repeat' not found
E006E Variable not found
E006F Syntax error
E0070 Invalid macro name
E0071 Redefined macro
E0072 Too many operands present
E0073 No operand is required
E0074 Label misplaced
E0075 Expression cannot be terminated by an operator
E0076 Negative value of operand2 is not allowed
E0077 Operand2 exceeds the max value, truncated from MSB
E0078 Negative value of operand1 is not allowed
E0079 Operand1 exceeds the max value, truncated from MSB
E007A Operand1 try to access the unimplemented SRAM
E007B Operand1 try to access the unimplemented SFR
E007C Try to access the unimplemented bit of SFR
E007D Operand2 points out of the implemented SRAM size
E007E Operand2 try to access the unimplemented SFR
E007F Encounter a drifting label
E0080 Label cannot be redefined
E0081 Symbol is taken as a label previously
E0082 Initialized local variable cannot be redefined as label
E0083 String should be closed by the paired-double quotes
E0084 More than one 'default' presents in the 'switch' block
E0085 Is used as a macro name
E0086 Instruction jumps across the page boundary
E0087 Loading failure of instruction table
E0088 Unknown list option encountered
E0089 Option setting not found
E008A Too many options encountered
E008B Unrecognized option

| | |
|---|---|
| E008C | Profile not found |
| E008D | IC reset is not allowed |
| E008E | Cannot use reserved memory |
| E008F | Overrun the program ROM size |
| E0090 | Encounter occupied code space |
| E0091 | Overrun the data ROM size |
| E0092 | The expression of start address not found |
| E0093 | Overlaid memory space is not allowed |
| E0094 | Data file is greater than data ROM size |
| E0095 | Cannot include data file before the processor is declared |
| E0096 | Cannot define data before the processor declaration |
| E0097 | Unrecognized statement |
| E0098 | Missing the paired double quote “” |
| E0099 | Only one of 'case','default','endsw','ends' statement allowed |
| E009A | Program is terminated with an opened 'cblock'-scope |
| E009B | Program is terminated in an unfinished macro block |
| E009C | Macro expansion exceeds the maximum expansion depth |
| E009D | Expression of max macro depth not found |
| E009E | Maximum depth of macro expansion must be greater than 0 |
| E009F | Current loops exceed the maximum loop count |
| E00A0 | Is an invalid name of defined symbol |
| E00A1 | Program is terminated without 'end' |
| E00A2 | Processor type not found ! |
| E00A3 | Loading failure ! |
| E00A4 | Is not supported ! |
| E00A5 | Expression is out of word scale 0xffff |
| E00A6 | Expression is out of byte scale 0xff |
| E00A7 | Expression/char is out of nibble scale 0xf |
| E00A8 | Empty 'dn' |
| E00A9 | Object code exceeds the ROM word size (0x3ff) |
| E00AA | Cannot use configuration block |
| E00AB | Encounters an invalid format profile |
| E00AC | Config block file not found ! |
| E00AD | Config block file is created by another ic type ! |
| E00AE | Config block exceeds the size of reserved memory block ! |
| E00AF | Missing the operand ! |
| E00B0 | Try to read the write-only bit of SFR |

E00B1 Try to write the read-only bit(%d) of SFR
E00B2 Operand try to read the write-only SFR
E00B3 Operand try to write the read-only SFR
E00B4 Encounters an invalid macro argument
E00B5 Step-zero is not allowed
E00B6 Label cannot be placed here
E00B7 Empty 'df'
E00B8 Expression is out of 5-bits scale 0x1f
E00B9 Macro call needs argument(s)
E00BA Reserved memory doesn't exist
E00BB Space for the confirm-info is occupied at address
E00BC Cannot call a macro behind the conditional branch
E00BD Mld4 format is not supported
E00BE Mld4 exceeds the addressing range of DPR

## B.2 Warnings

W0000 Macro expansion is turned on already
W0001 Macro expansion is turned off already
W0002 User defined warning message
W0003 Operand2 not found,default value is used
W0004 Operand not found,default value is used
W0005 Re-defined variable
W0006 Ignored, for invalid expression, Line/Page
W0007 Operand2 accesses the partial implemented SFR
W0008 Operand1 accesses the partial implemented SFR
W0009 ORG without specified PC value is ignored
W000A Title string not found
W000B Too many title strings encountered
W000C Sub-title string not found
W000D Too many sub-title strings encountered
W000E String should be closed by a double quotes
W000F Reset title
W0010 Reset subtitle
W0011 The changing of case-sensitive may cause incorrect result
W0012 Object code exceeds the ROM word boundary (0x3ff)
W0013 Break by user
W0014 Confirm-information is not inserted

# Glossary

To provide a common frame of reference, this glossary defines the terms that are used in this document. This glossary contains definitions for the terms used in the AMCA.

## C.1 Terms

**Alpha Character**

Alpha characters are those characters, regardless of case, that are normally contained in the alphabet: (a, b, …, z, A, B, …, Z).

**ALPHA MCU**

ALPHA MCU refers to the AM4X, AM8X micro-controller families.

**Alphanumeric**

Alphanumeric characters include alpha characters and numbers: (0,1, …, 9).

**Application**

A set of software and hardware developed by the user, usually designed to be a product controlled by a ALPHA micro-controller.

**Assemble**

The act of executing the AMCA macro assembler to translate source code to machine code.

**Assembler Source Code**

A text file that is processed by an assembler to produce a one-to-one correspondence between assembler instructions and ALPHA MCU machine code.

**Binary File**

An AMCA single executable output.

**Build**

A function that recompiles all the source files for an application.

**Control Directives**

Control directives permit sections of conditionally assembled code.

**Data Directives**

Data Directives are those that control the allocation of memory and provide a way to refer to data items symbolically, that is, by meaningful names.

**Data RAM**

General purpose file registers from RAM on the MCU device being emulated.

**Directives**

Directives provide control of the assembler's operation by telling AMCA how to treat mnemonics, define data, and format the listing file. Directives make coding easier and provide custom output according to

specific needs.

**Expressions**

Expressions are used in the operand field of the source line and may contain constants, symbols, or any combination of constants and symbols separated by arithmetic operators.

**Identifier**

A function or variable name.

**Initialized Data**

Data which is defined with an initial value. In C, int myVar=5; defines a variable which will reside in an initialized data section.

**Listing Directives**

Listing Directives are those directives that control the AMCA listing file format. They allow the specification of base-numbering system, reserved memory access and other listing control.

**Listing File**

A listing file is an ASCII text file that shows the machine code generated for each assembly instruction, AMCA directive, or macro encountered in a source file.

**Local Label**

A local label is one that is defined with the `LOCAL` directive. These labels are particular to a given instance of the macro's instantiation. In other words, the symbols and labels that are declared as local are purged from the symbol table when the `ENDM` macro is encountered.

**Macro**

A macro is a collection of assembler instructions that are included in the assembly code when the macro name is encountered in the source code. Macros must be defined before they are used; forward references to macros are not allowed. All statements following the `MACRO` directive are part of the macro definition. Labels used within the macro must be local to the macro so the macro can be called repetitively.

**Macro Directives**

These directives control the execution and data allocation within macro body definitions.

**Mnemonics**

These are instructions that are translated directly into machine code. These are used to perform arithmetic and logical operations on data residing in program or data memory of a micro-controller. They also have the ability to move data in and out of registers and memory as well as change the flow of program execution. Also referred to as **Opcodes**.

**AMCA**

ALPHA Microelectronics Corporation's MCU assembler.

**Nesting Depth**

Macros can be nested to 16 levels deep (default). Maximum depth is 256.

**Object Code**

The machine code that is produced from the source code after it is processed by an assembler. This code will be the memory-resident code that will run on the ALPHA MCU in the user's application. Object code is stored in a binary file.

**Operators**

Operators are arithmetic symbols, like the plus sign '+' and the minus sign '-', that are used when forming well-defined expressions. Each operator has an assigned precedence.

**PC**

Any IBM® or compatible Personal Computer.
machine.

**PC Host**

The computer running Windows XP/2000 or Windows 95/98.

**Precedence**

Precedence is the concept that some elements of an expression get evaluated before others. Operators of the same precedence are evaluated from left to right.

**Program Memory**

Memory in the emulator or simulator containing the downloaded target application firmware.

**Project**

A set of source files and instructions to build the object code for an application.

**Radix**

Radix is the base-numbering system that the assembler uses when evaluating expressions. The default radix is hexadecimal (base 16). You can change the default radix and override the default radix with certain radix override operators.

**RAM**

Random Access Memory (Data Memory).

**Raw Data**

The binary representation of code or data.

**Recursion**

This is the concept that a macro, having been defined, can call itself. Great care should be taken when writing recursive macros; it is easy to get caught in an infinite loop where there will be no exit from the recursion.

**ROM**

Read-only Memory (Program Memory).

**Shell**

The AMCA shell is a prompted input interface to the macro assembler. There is an AMCA shell for the Windows version.

**Source Code**

Source code consists of ALPHA MCU instructions and AMCA directives and macros that will be translated into machine code. This code is suitable for use by an ALPHA development system product like AMCIDE.

**Source File**

The ASCII text file of ALPHA MCU instructions and AMCA directives and macros (source code) that will be translated into machine code. It is an ASCII file that can be created using any ASCII text editor.

**Stack**

An area in data memory where function arguments, return values, local variables, and return addresses are stored.

**Symbol**

A symbol is a general purpose mechanism for describing the various pieces which comprise a program. These pieces include function names, variable names, file names, macro names, etc.

**Uninitialized Data**

Data which is defined without an initial value. In C, int myVar.

***NOTES:***

Information contained in this publication regarding device applications and the like is intended for suggestion only and may be superseded by updates. No representation or warranty is given and no liability is assumed by ALPHA Microelectronics Corporation with respect to the accuracy or use of such information, or infringement of patents or other intellectual property rights arising from such use or otherwise. Use of ALPHA's products as critical components in life support systems is not authorized except with express written approval by ALPHA. No licenses are conveyed, implicitly or otherwise, under any intellectual property rights. The ALPHA logo and name are registered trademarks of ALPHA Microelectronics Corporation and other countries. All rights reserved. All other trademarks mentioned herein are the property of their respective companies.

2003 ALPHA Microelectronics Corporation.

All rights reserved. © 2003 ALPHA Microelectronics Corporation. Published in the TAIWAN.